1. 理解格式文件，以wav声音为例。

在QQ群中的声音文件523hz8k8bit.wav，是一个wav声音文件，包含1秒时间的523hz音乐dou的声音。数字化：8000hz采样率，每个样本8bit。

参考网页：<https://www.cnblogs.com/LexMoon/p/wave-c.html>

1）、解释出本文件的文件头内容。

2）、读出文件头后的50个采样点的数据，画出前50个采样点的图形。（将采样点数值/10, 结果最大是26，在【0，0】【50，26】的屏幕上显示\*表示采样点。）

\* \* \*\*

\* \* \* \* \* \*

\* \* \* \*

\* \* \*

\* \* \* \*

\* \*

\* \* \*

\* \* \*

\* \*

\* \* \*

\* \*

\* \* \* \*

\* \* \*

\*\* \*\* \*\*\*

3）、显示曲线显示的动画效果。向左移动。

代码框架：

for(k=0;k<DATA\_LEN;k++)

{

printf("\x1b[H\x1b[2J"); // 清屏

// 此处显示曲线

usleep(100000l); //延时100ms

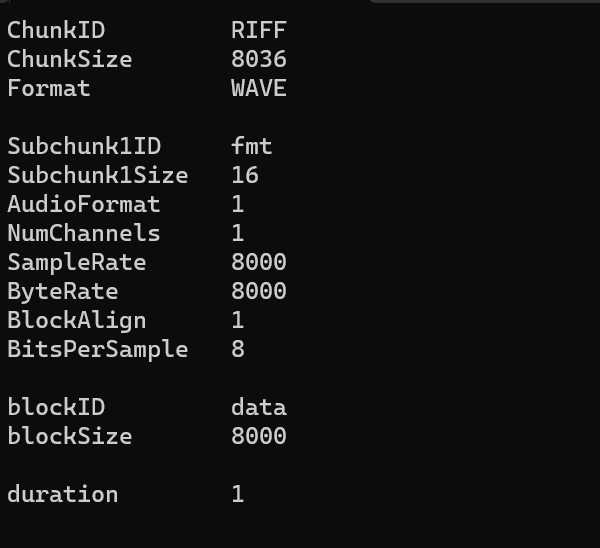
// 此处向左移动一个单位

 }

<https://blog.csdn.net/u014608280/article/details/84583961>

<https://blog.csdn.net/whatday/article/details/108942583>

（1）



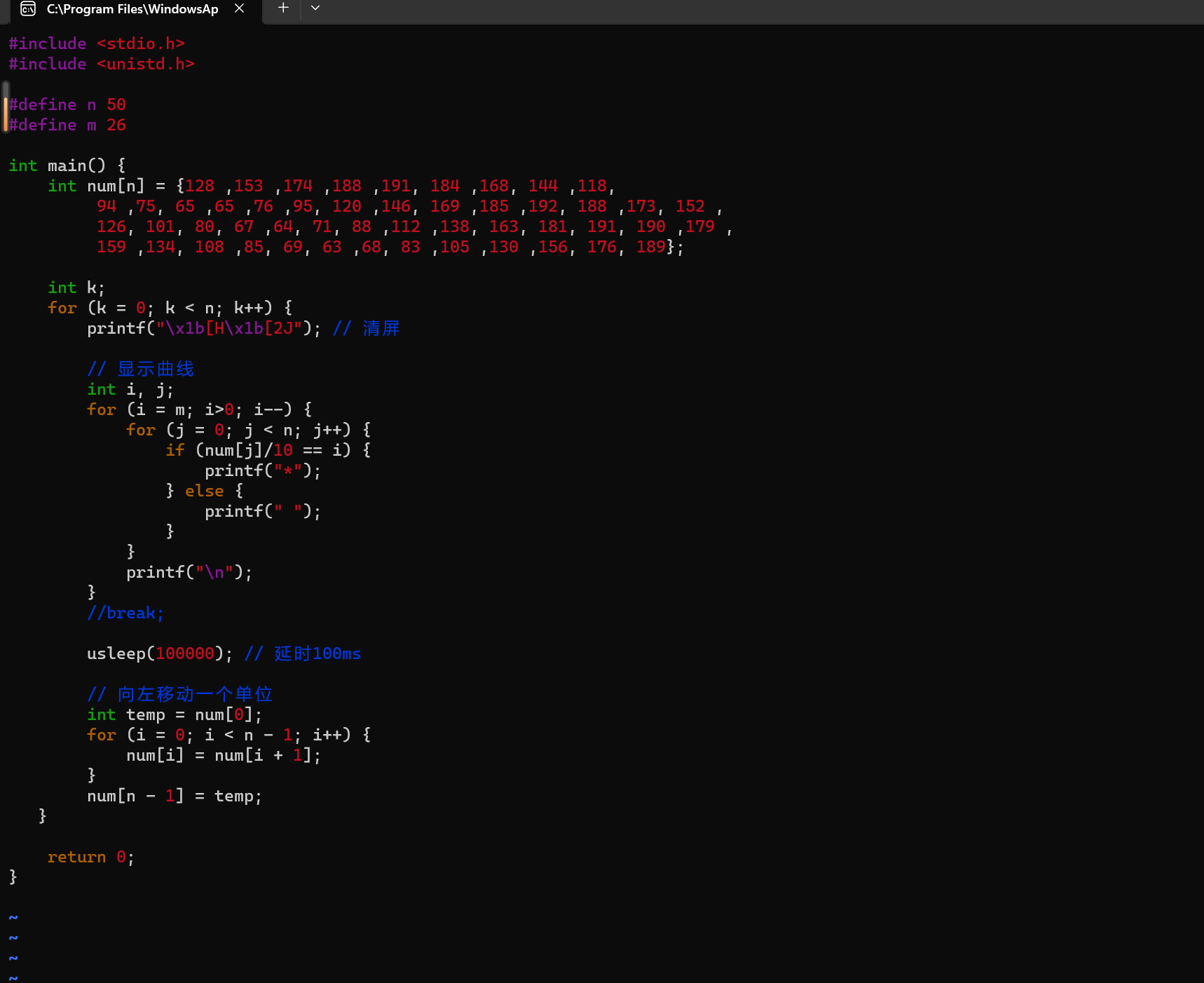
解释：

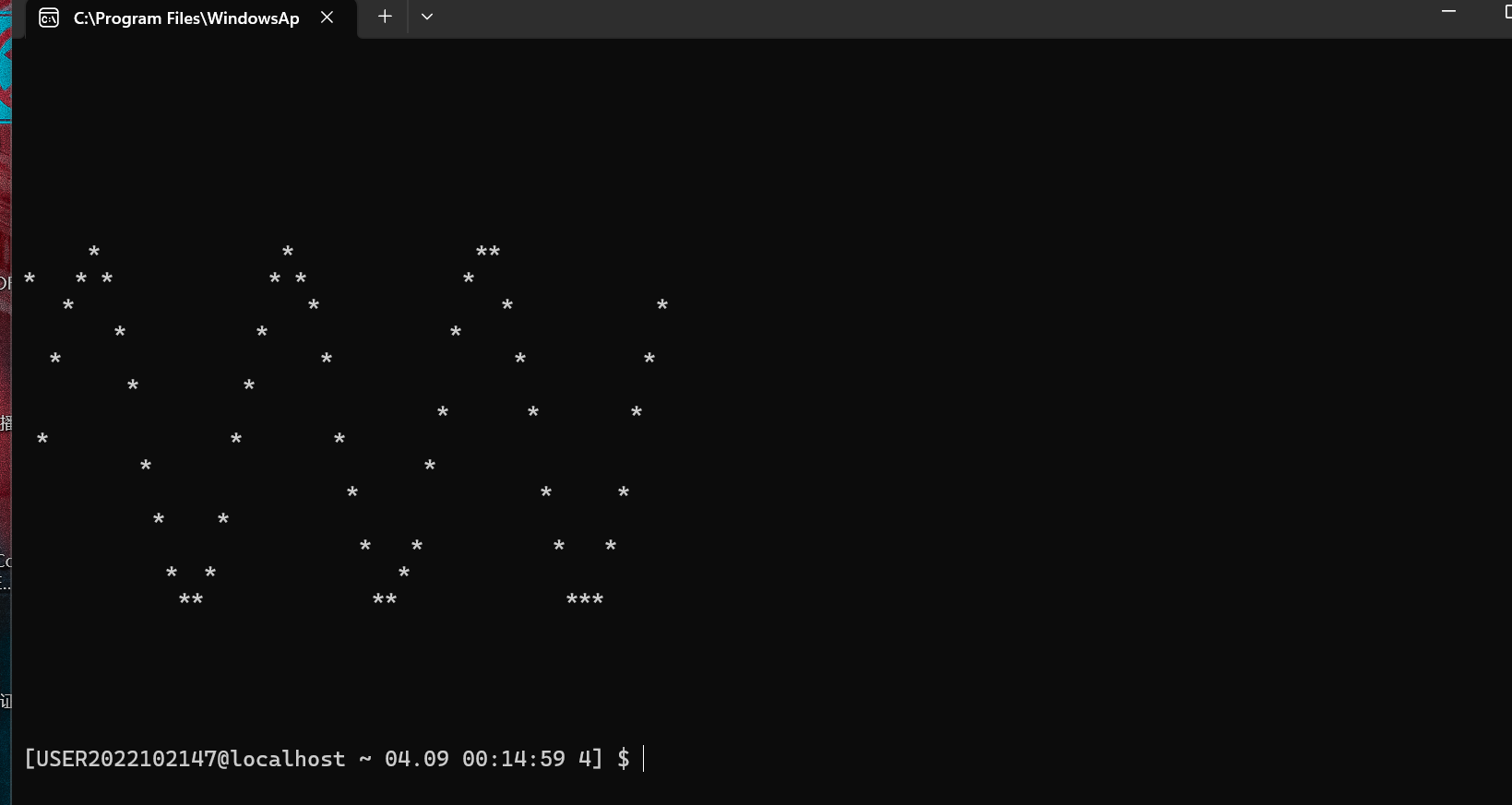
* **ChunkID**: 标识这是一个RIFF文件，即以"RIFF"四个字母作为文件标识符。
* **ChunkSize**: 表示文件大小，不包括ChunkID和ChunkSize字段本身的大小，以及该字段之后的WAV文件大小。
* **Format**: 指定文件格式，这里是WAVE格式。
* **Subchunk1ID**: 指定子块的ID，即fmt子块。
* **Subchunk1Size**: 指定fmt子块的大小，通常为16字节。
* **AudioFormat**: 指定音频数据的编码格式，1表示PCM，即脉冲编码调制。
* **NumChannels**: 指定声道数，这里为1，表示单声道。
* **SampleRate**: 指定采样率，即每秒钟的采样次数，这里为8000Hz。
* **ByteRate**: 指定比特率，即每秒钟的数据量，这里为8000字节。
* **BlockAlign**: 指定块对齐大小，即每个采样点占用的字节数，这里为1字节。
* **BitsPerSample**: 指定每个采样点的位数，这里为8位。
* **blockID**: 指定数据块的ID，即data子块。
* **blockSize**: 指定数据块的大小，即音频数据的大小，这里为8000字节，对应1秒的音频数据。
* **duration**: 指定音频的持续时间，这里为1秒。

（2）数据如下



（3）





2. 了解一下linux系统

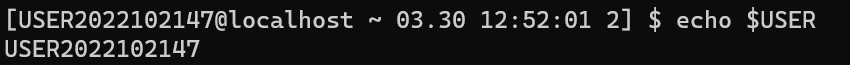
1) 查一下你是谁？

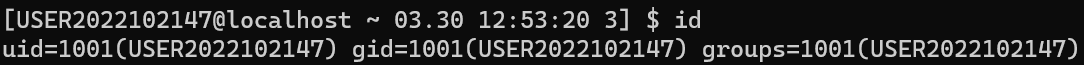
$ echo $USER

$ id

uid=1000(*USER20xx057550*) gid=1000(wang) groups=1000(wang)

$



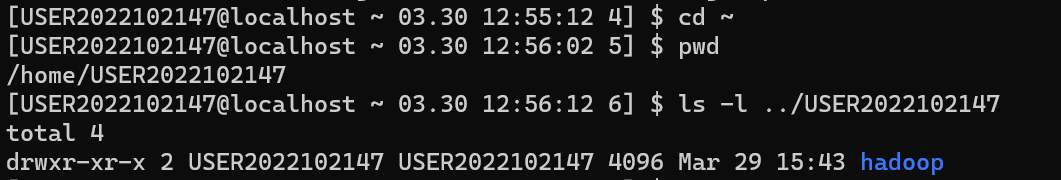


2) 你现在是在哪里？当前位置有哪些权限？

$ cd ~

$ pwd

$ ls -l ../ *USER20xx057550*



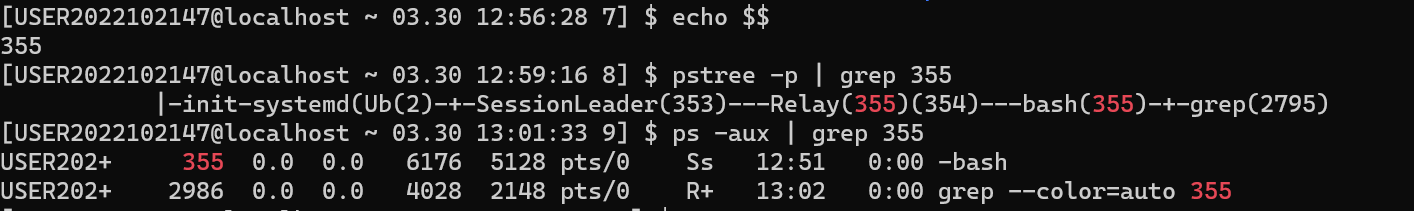
3)、你现在什么环境下(当前拥有终端【键盘与显示器】的进程)？找出你的环境下，登录shell的PID

**$ echo $$**

*xxxx*

**$ pstree -p | grep** *xxxx*

$ ps -aux | grep xxxx

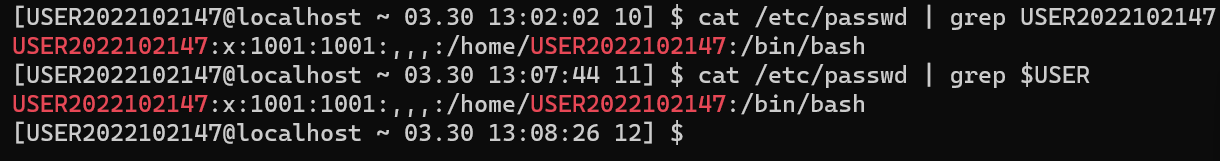


4)、你是从哪里来？

$ cat /etc/passwd | grep *USER20xx057550*

*USER20xx057550*:x:1000:1000: *USER20xx057550*,220,,:/home/ *USER20xx057550*:/bin/bash

$ cat /etc/passwd | grep $USER



3．（当前进程相关的ID属性）

#include <stdio.h>

#include <unistd.h>

int main()

{

printf("pid:%d, ppid:%d, uid:%d, gid:%d\n", getpid(), getppid(), getuid(), getgid());

return 0;

}

将以上程序命名为testid.c，在你的linux环境中运行以下命令，给出命令结果，并解释命令结果。

$ cd ~ # 转到工作目录

$ vi testid.c #输入源程序

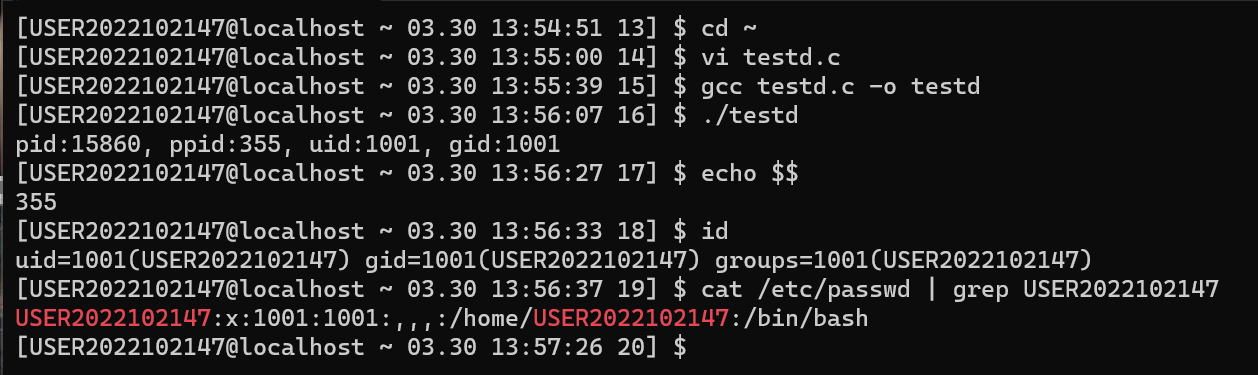
$ gcc testid.c -o testid #编译成可运行文件

$ ./testid # 运行程序

$ echo $$ # 显示当前进程ID

$ id #显示当前用户ID

$ cat /etc/passwd | grep *USER20xx057550*

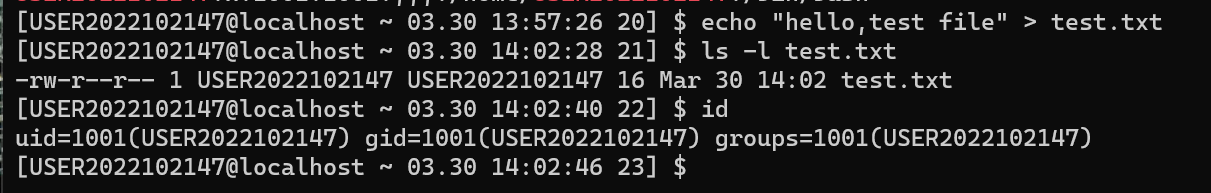


4、创建你自己的文件，看一下它的文件属性？（用户新创建文件时，新文件的相关属性）

$ echo “hello , test file “ > test.txt

$ ls -l test.txt

$ id



5、linux信号测试（理解信号， signal函数）

<https://blog.csdn.net/ljianhui/article/details/10128731>

<http://gityuan.com/2015/12/20/signal/>

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

void mysignal(int sig)

{

printf("\n I got signal %d \n", sig ); // 显示捕获的信号

// 恢复终端中断信号SIGINT的默认行为

// (void) signal(SIGINT, SIG\_DFL); // 注解1

}

int main()

{

//改变终端中断信号SIGINT的默认行为，使之执行 mysignal函数

//而不是终止程序的执行

// signal(SIGINT, mysignal); // i注解 2

while(1)

{

printf("Hello World!\n");

sleep(10); // 睡眠10秒

}

return 0;

}

以上程序命令为：testsignal.c

vi testsignal.c

gcc testsignal.c -o testsignal

./testsignal

# 按 CTRL+C 组合键。

#（该组合键会向当前进程发信号 SIGINT， 信号的默认动作是终止进程）

cp testsignal.c testsignal1.c

vi testsignal1.c

# 取消 “注解 2” 行最前面的注解，再编译

gcc testsignal1.c -o testsignal1

./testsignal1

# 按 CTRL+C 组合键。

# （该组合键会向当前进程发信号 SIGINT， 信号的默认动作是终止进程）

# 按 CTRL+C 组合键。

# 按 CTRL+C 组合键。

# 按 CTRL+\ 组合键。 （该组合键会向当前进程发信号 SIGQuIT， 信号的默认动作是终止进程）

cp testsignal1.c testsignal2.c

vi testsignal2.c

# 取消 “注解 1” 行最前面的注解，再编译

gcc testsignal2.c -o testsignal2

./testsignal2 &

pstree $$ -p

bash(4766)─┬─pstree(4904)

└─testsignal2(4903)

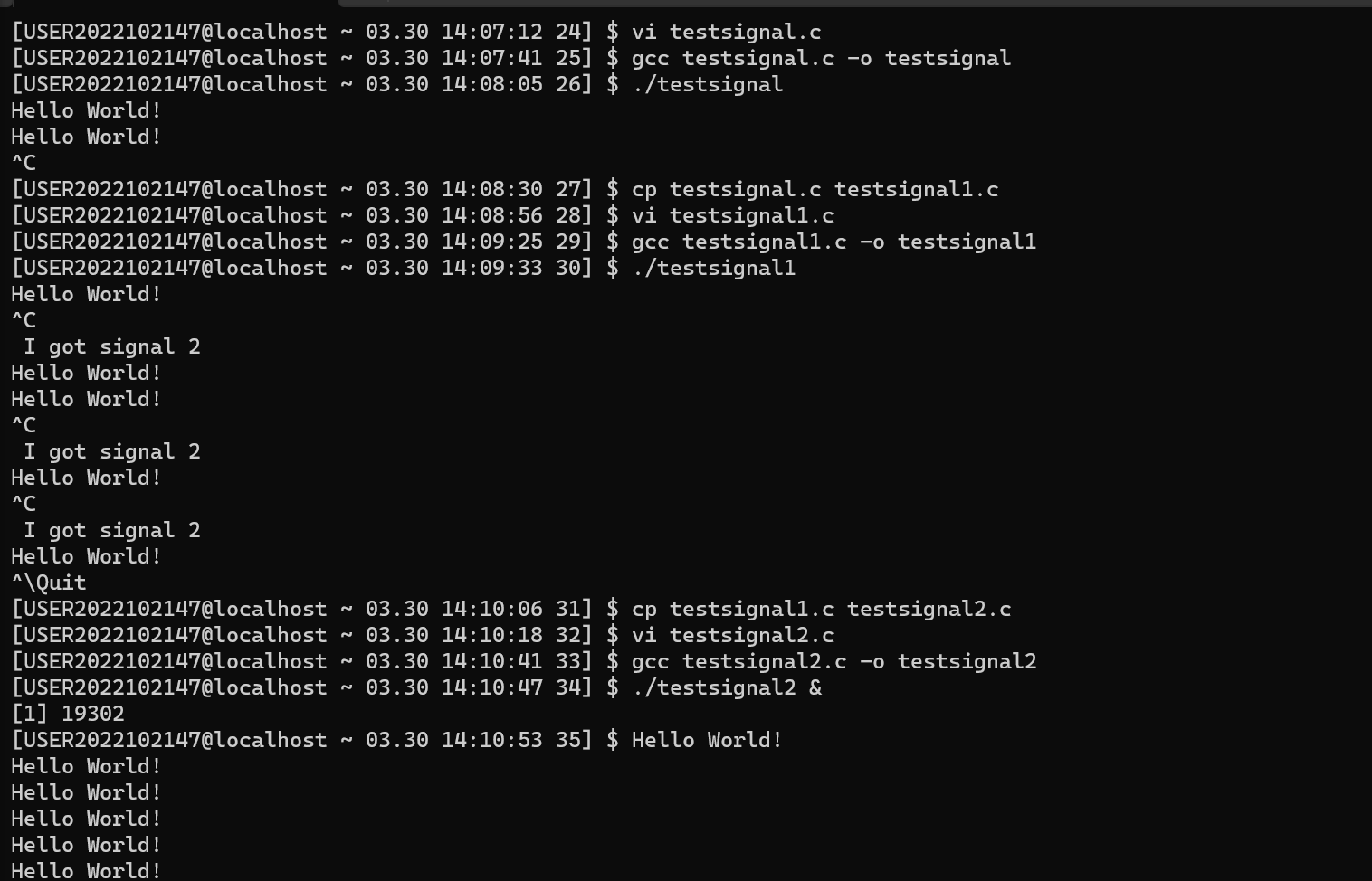
kill -SIGINT 4903 # 向进程发送 SIGINT 信号

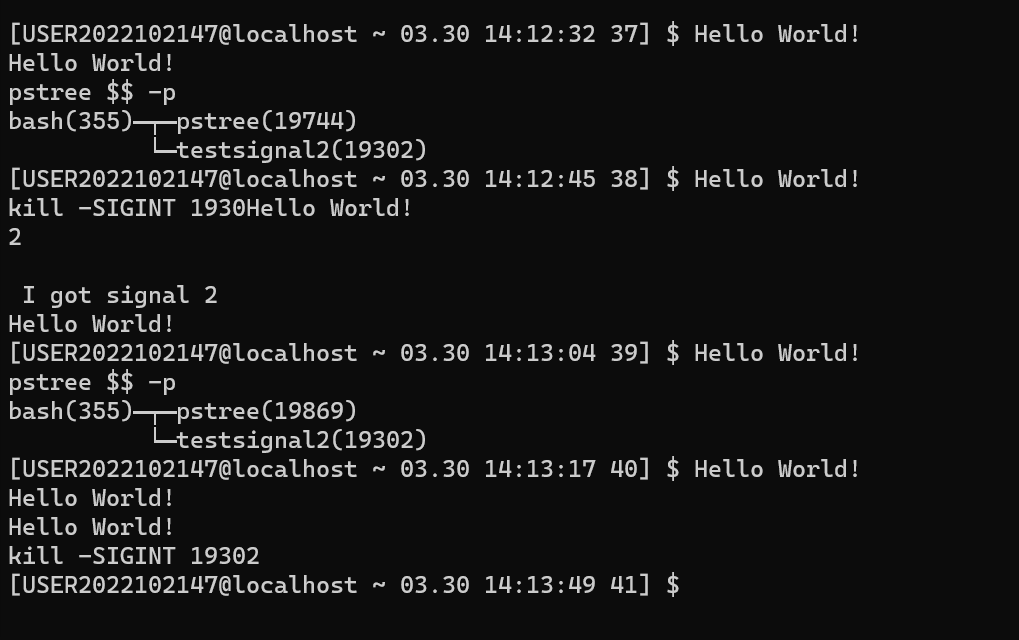
pstree $$ -p

kill -SIGINT 4903

kill -SIGINT 19302

pstree $$ -p





6、理解pause.

<https://blog.csdn.net/zgdzjzz/article/details/98513180>

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

void mysignal(int sig)

{

printf("\n I got signal %d \n", sig ); // 显示捕获的信号

}

int main()

{

//改变终端中断信号SIGUSR1的默认行为，使之执行 mysignal函数

signal(SIGUSR1, mysignal);

while(1)

{

printf("Hello World!\n");

pause(); // 让自己睡眠，等信号发生

}

return 0;

}

以上程序命令为：testpause.c

vi testpause.c

gcc testpause.c -o testpause

./testpause &

pstree $$ -p

bash(4766)─┬─pstree(4904)

└─testpause(4903)

kill -SIGUSR1 21021

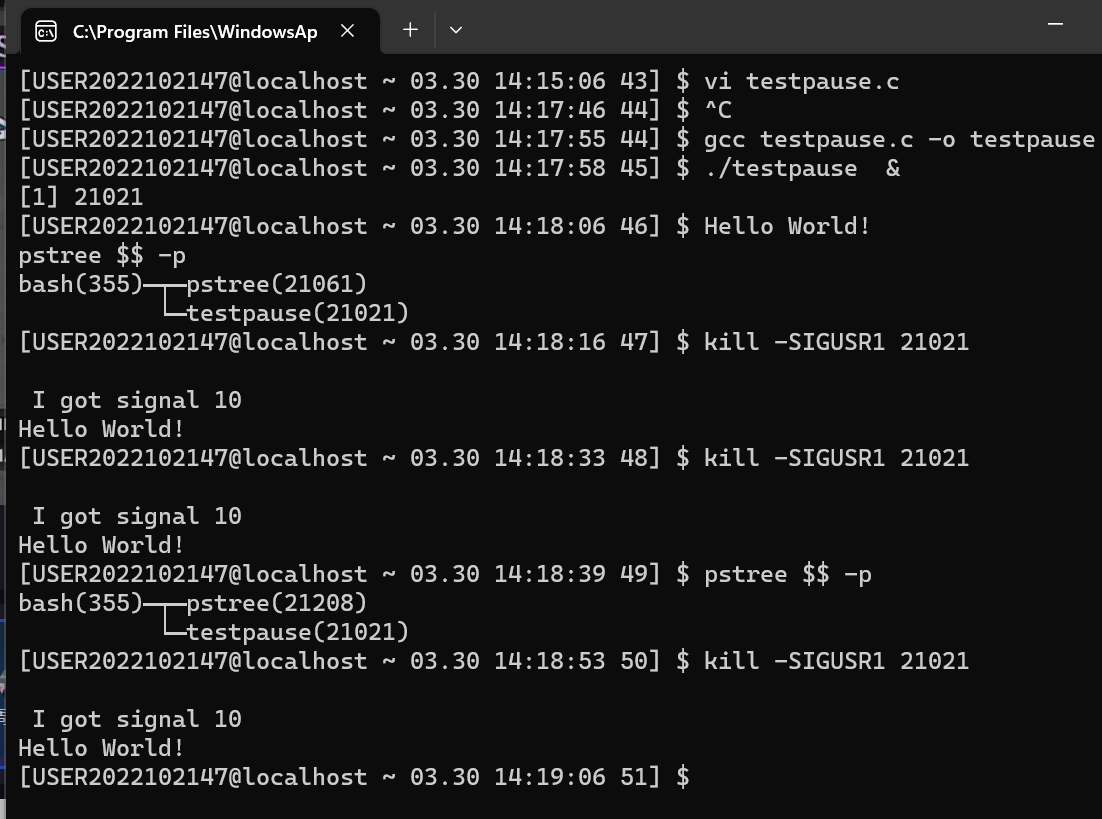
pstree $$ -p

kill -SIGUSR1 4903

pstree $$ -p

kill -SIGINT 4903

pstree $$ -p



7、程序如何获得环境的参数（命令行参数及环境参数）

#include <stdio.h>

#include <unistd.h>

#include<signal.h>

extern char\*\* environ;

void handler(int n) //自定义函数，用于收到信号, 继续运行程序

{

printf("get signal USR1， test prg continue \n");

}

int main(int argc,char \*argv[])

{

int i;

signal(SIGUSR1,handler); // 设置信号SIGUSR1 处理程序

for (i=0; argv[i]!=NULL; ++i)

printf("argv[%d]=%s\n", i,argv[i]);

for (i=0; environ[i]!=NULL; ++i)

printf("environ[%d]=%s\n",i, environ[i]);

printf("test prg, %s, pid:%d, ppid:%d, euid:%d\n", argv[0],getpid(), getppid(), geteuid());

// sleep(15); //进程睡眠15秒

pause(); // 让自己睡眠 （通过 信号 可以唤醒）

printf("test prg end. \n");

return(0); // 返回，即结束当前进程

}

以上程序命名为：testp.c

vi testp.c

gcc testp.c -o testp

echo $$

./testp one two2 three3 & # &表示后台运行，运行后，请多按一个回车。

pstree $$ -p # 以下红色部分为系统显示，记住黄底数字

bash(3517)─┬─pstree(3880)

└─testp(3877)

kill -USR1 3877 #黄色数字为你的环境下的进程PID

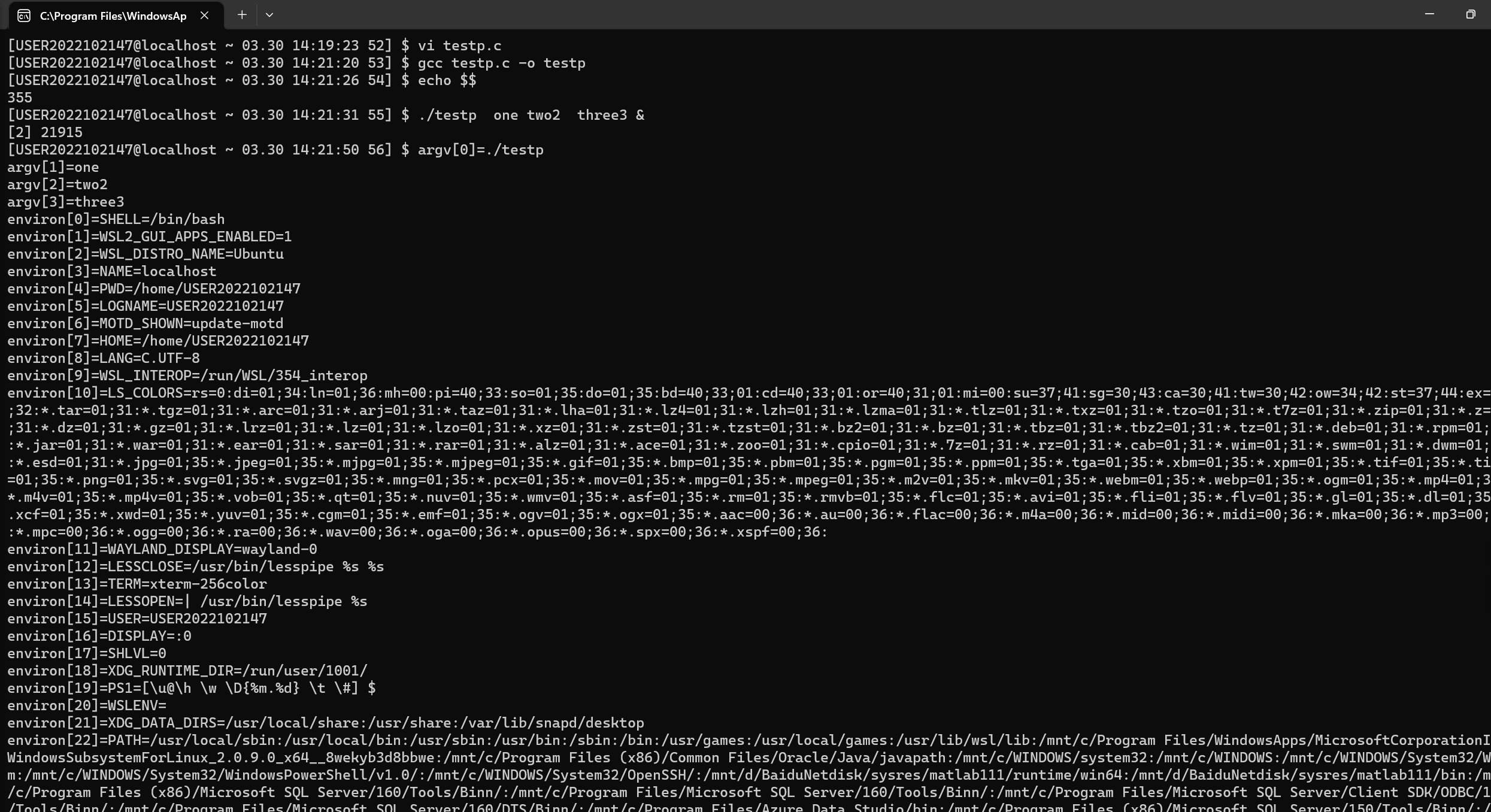
./testp one two2 & # &表示后台运行，运行后，请多按一个回车。

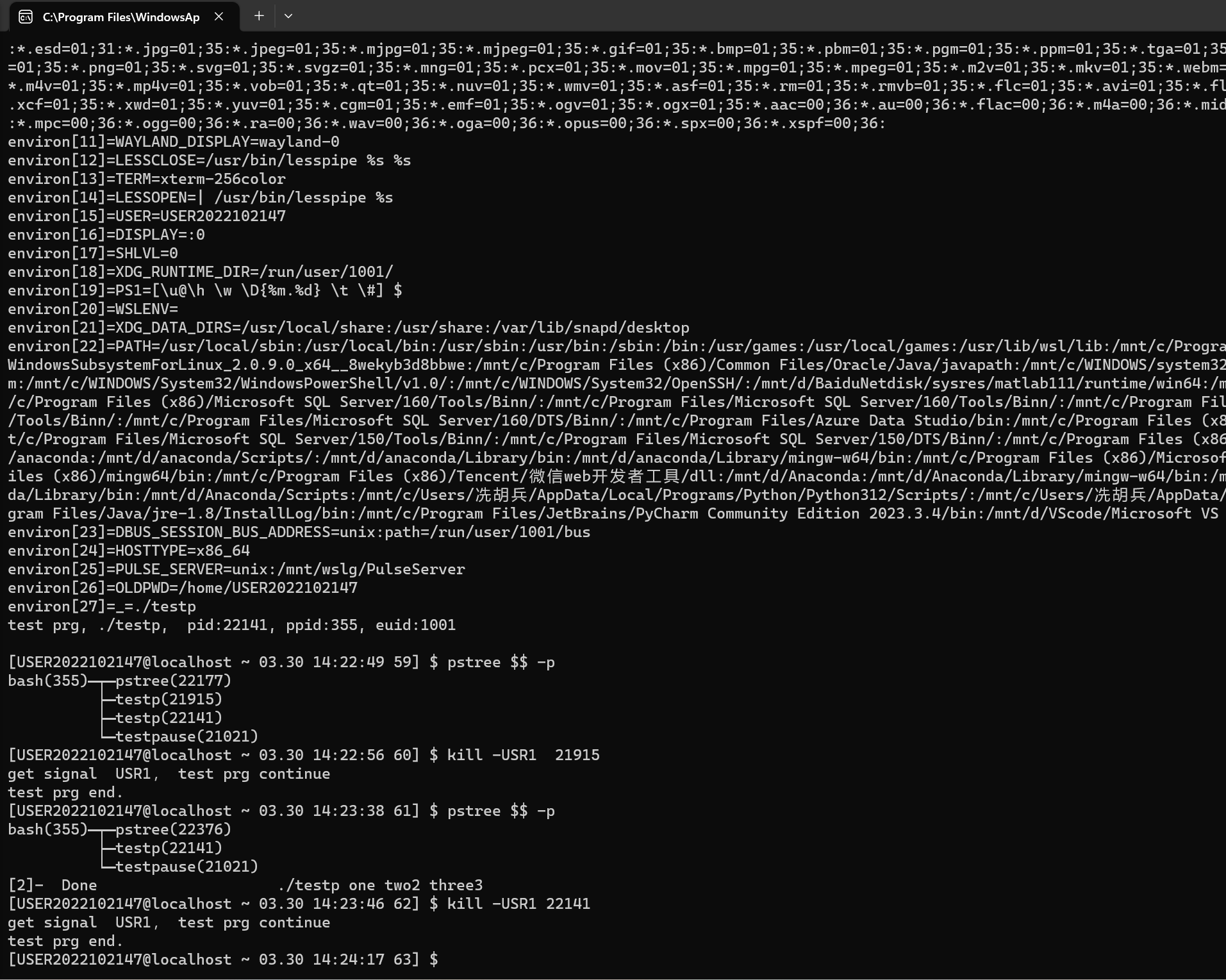
pstree $$ -p # 以下红色部分为系统显示，记住黄底数字

bash(3517)─┬─pstree(3980)

└─testp(3911)

kill -USR1 3911 #黄色数字为你的环境下的进程PID





8、理解HTTP协议。

<https://www.cnblogs.com/laixiangran/p/8777579.html> 前端开发必备之Chrome开发者工具（上篇）

<https://www.cnblogs.com/jycboy/p/http1.html> 带新手走进神秘的HTTP协议

<https://blog.csdn.net/weixin_38087538/article/details/82838762> Http协议详解(深入理解)

<https://blog.51cto.com/u_15326983/3332582> 用Chrome开发者工具查看HTTP请求与响应

用chrome的开发者工具，以网址 [www.jnu.edu.cn](http://www.jnu.edu.cn) 为例，给出http协议的请求与响应内容。

9、学习git clone的用法，及c语言编写的基于http协议的WEB服务器程序的简单原型。

（本次作业用c语言实现最简单功能的web 服务器）

如果git clone 有问题， 也可以直接在：<https://github.com/shenfeng/tiny-web-server.git>

上下载。

安装git, 如果前面作业中已正确安装git，可忽略以下三个命令。

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install git

1)、从git上克隆项目

$ cd ~

$ git clone https://github.com/shenfeng/tiny-web-server.git

Cloning into 'tiny-web-server'...

remote: Enumerating objects: 80, done.

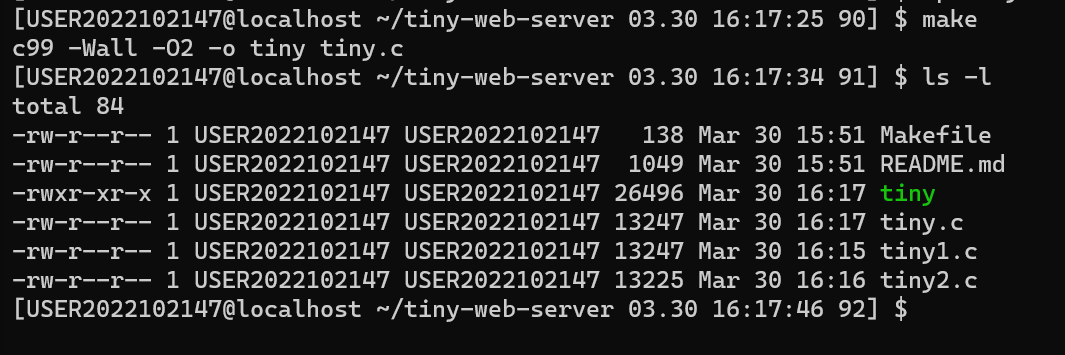
remote: Total 80 (delta 0), reused 0 (delta 0), pack-reused 80

Unpacking objects: 100% (80/80), 15.44 KiB | 166.00 KiB/s, done.

2)、编译工程

$ cd tiny-web-server

$ make



3)、确认编译成功

$ ls -l

total 44

-rw-rw-r-- 1 ubuntu ubuntu 138 Apr 15 04:05 Makefile

-rw-rw-r-- 1 ubuntu ubuntu 1049 Apr 15 04:05 README.md

-rw-rw-r-- 1 ubuntu ubuntu 135 Apr 15 04:09 index.html

-rwxrwxr-x 1 ubuntu ubuntu 14328 Apr 15 04:05 tiny

-rw-rw-r-- 1 ubuntu ubuntu 13225 Apr 15 04:05 tiny.c

4)、在当前项目的目录下创建一个叫 index.html 的文件

<!DOCTYPE html>

<html>

<head lang="en">

<meta charset="UTF-8">

<title></title>

</head>

<body>

tiny web server

</body>

</html>

5)、启动服务程序；

$./tiny 6666 &

1] 15654

listen on port 6666, fd is 3

child pid is 15656

child pid is 15657

child pid is 15658

child pid is 15659

child pid is 15660

child pid is 15661

child pid is 15662

child pid is 15663

child pid is 15664

child pid is 15665

6)、 在服务器上测试：

$ curl 127.0.0.1:6666

<html><head><style>body{font-family: monospace; font-size: 13px;}td {padding: 1.5px 6px;}</style></head><body><table>

<tr><td><a href="README.md">README.md</a></td><td>2023-04-03 21:34</td><td>1.0K</td></tr>

<tr><td><a href=".git/">.git/</a></td><td>2023-04-03 21:34</td><td>[DIR]</td></tr>

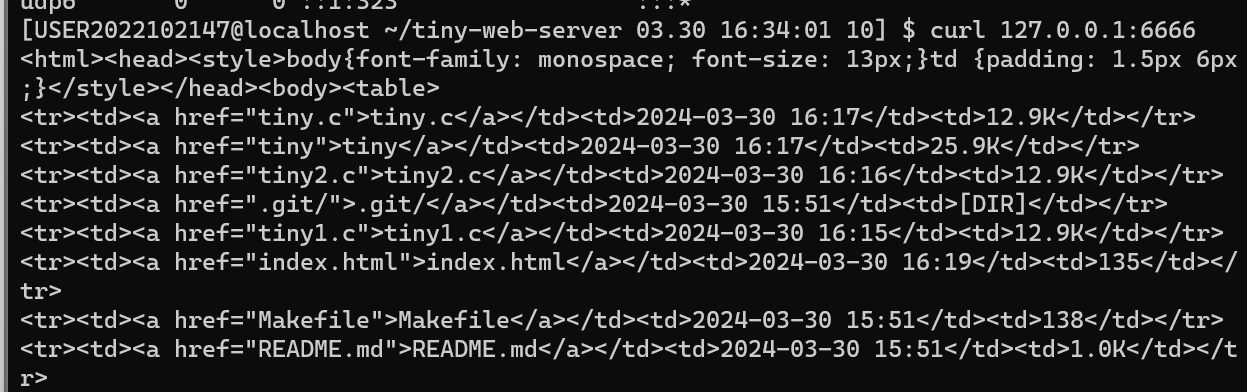
<tr><td><a href="tiny">tiny</a></td><td>2023-04-03 22:02</td><td>18.4K</td></tr>

<tr><td><a href="Makefile">Makefile</a></td><td>2023-04-03 21:34</td><td>138</td></tr>

<tr><td><a href="index.html">index.html</a></td><td>2023-04-03 21:37</td><td>135</td></tr>

<tr><td><a href="tiny.c">tiny.c</a></td><td>2023-04-03 22:02</td><td>13.0K</td></tr>

</table></body></html>



$ curl 127.0.0.1:6666/index.html

accept request, fd is 4, pid is 15658

offset: 135

127.0.0.1:37850 200 - index.html

<!DOCTYPE html>

<html>

<head lang="en">

<meta charset="UTF-8">

<title></title>

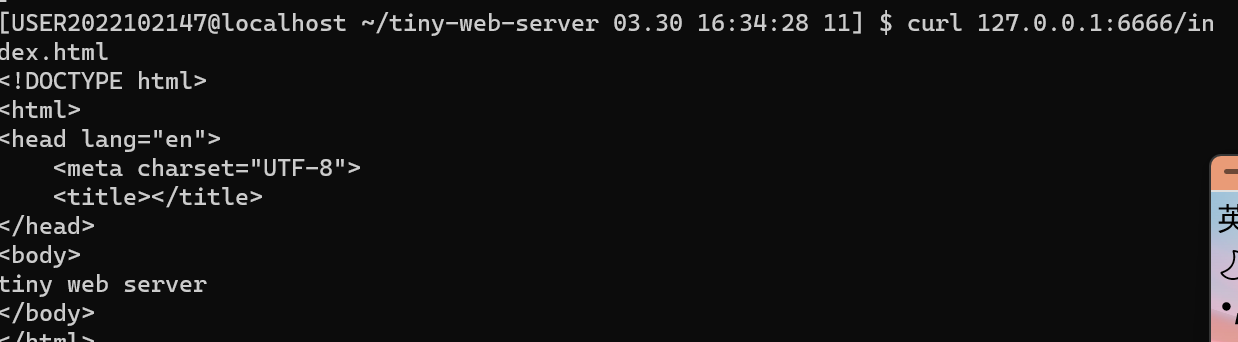
</head>

<body>

tiny web server

</body>

</html>



7)、在客户端测试：

$ ifconfig #查看IP地址

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.5.101 netmask 255.255.240.0 broadcast 192.168.255.255

inet6 fe80::215:5dff:fed0:aa01 prefixlen 64 scopeid 0x20<link>

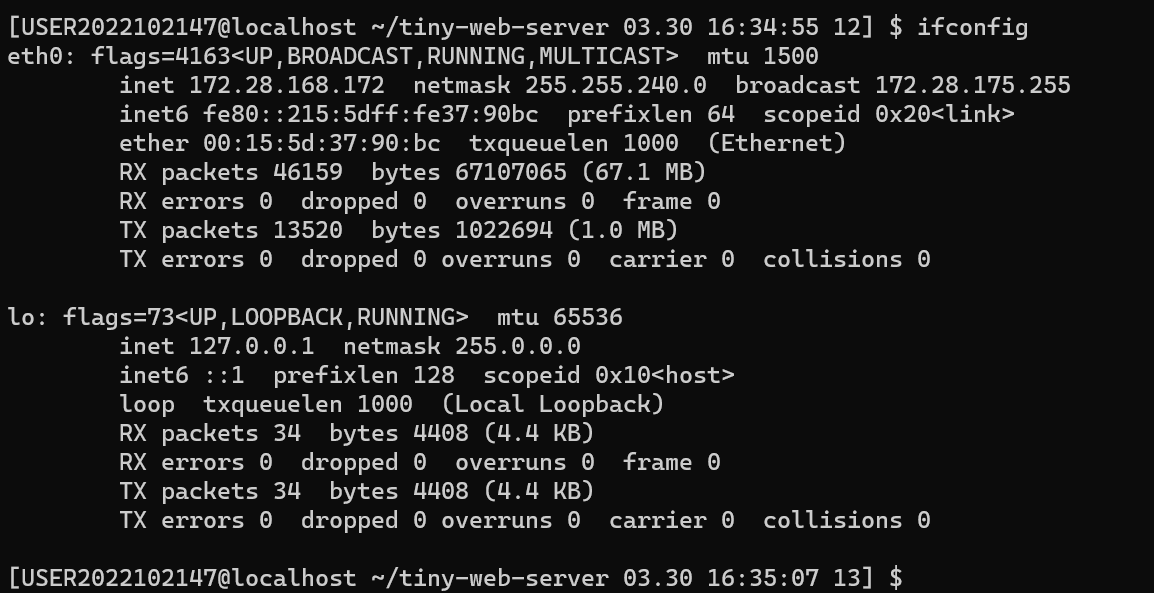
ether 00:15:5d:d0:aa:01 txqueuelen 1000 (Ethernet)

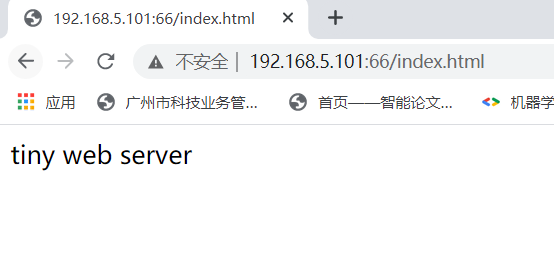
RX packets 14909 bytes 20096001 (20.0 MB)

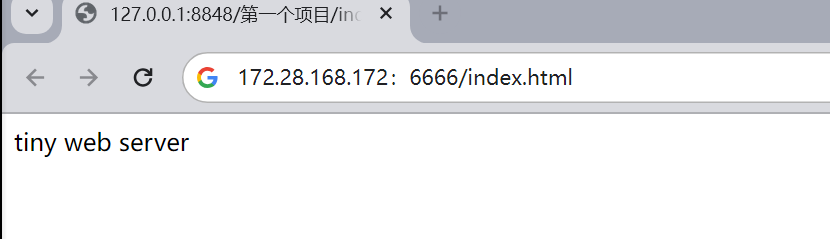
RX errors 0 dropped 0 overruns 0 frame 0

打开浏览器，输入URL：（IP地址为前面通过 ifconfig 查到的linux主机地址）

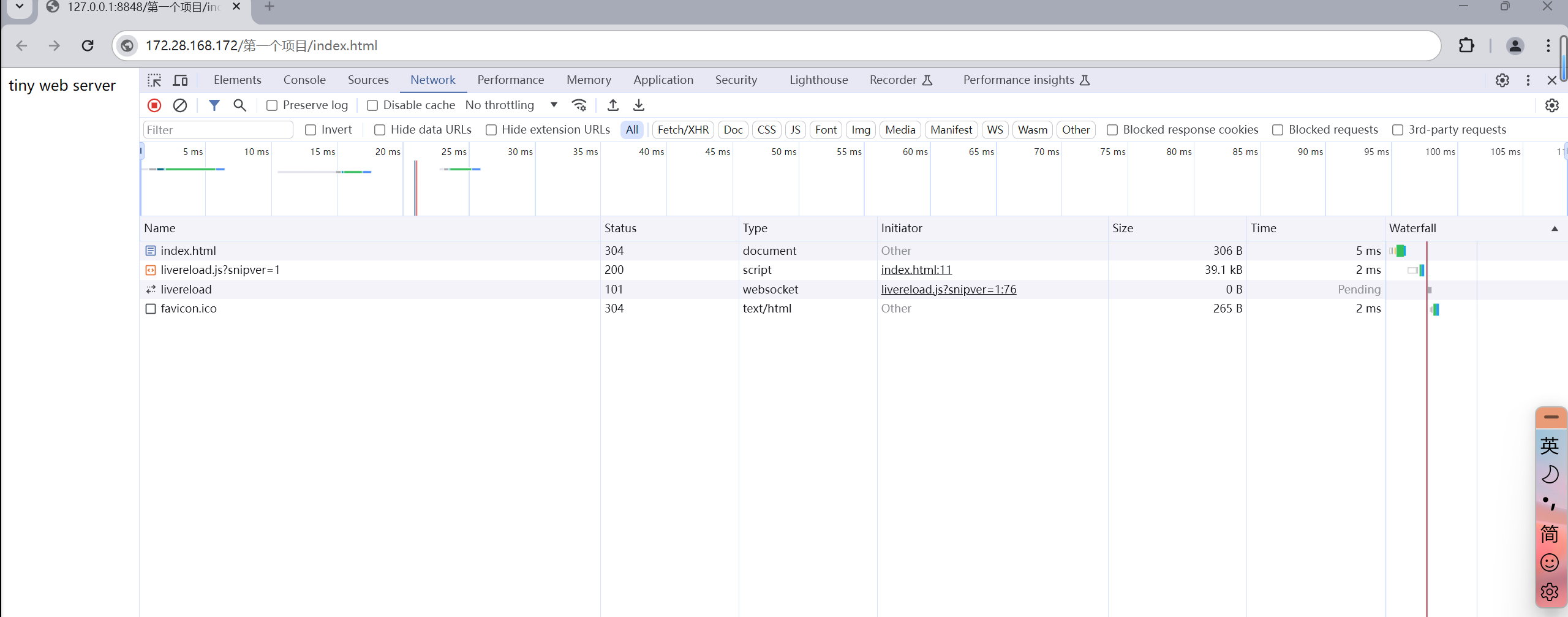
http://192.168.5.101:6666/index.html

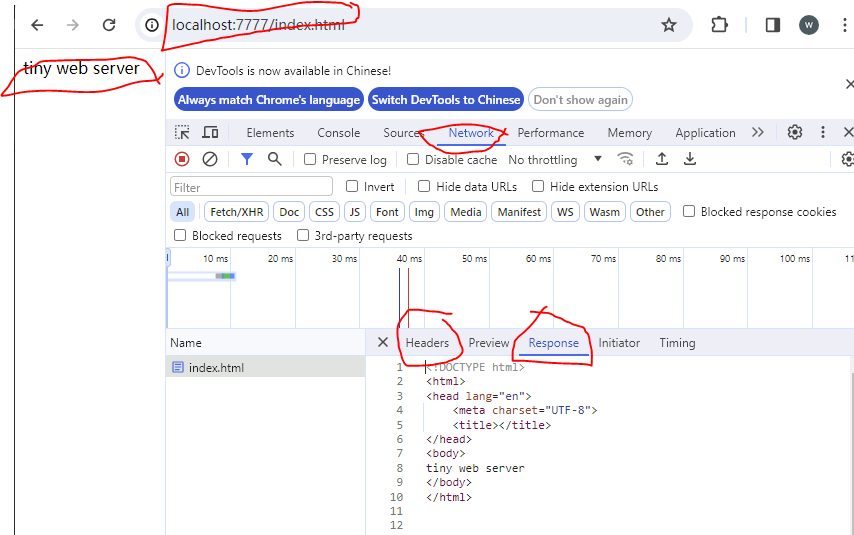






8）、用chrome开发者工具，观察http的请求与响应过程。





在你的环境下，测试上述过程。给出运行结果，并简单解释上述每条命令的功能。

tinyhttpd —— C 语言实现最简单的 HTTP 服务器(code)

<https://www.cnblogs.com/tureno/articles/6270329.html>

10、C语言实现http的web客户端（只实现了最基本的get命令）。

#include <stdarg.h>

#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <sys/socket.h>

#include <resolv.h>

#include <stdlib.h>

#define MAXBUF 1024

void error\_exit(char \*msg)

{

printf("error %s \n",msg);

exit(-1);

}

int main(int argc, char \*argv[])

{ int sockfd, bytes\_read;

struct sockaddr\_in dest;

char buffer[MAXBUF];

/\*---Make sure we have the right number of parameters---\*/

if ( argc != 3 )

error\_exit ("usage: http-client <IP-addr> <Port>\n");

if ( (sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0 )

error\_exit ("Socket");

/\*---Initialize server address/port struct---\*/

bzero(&dest, sizeof(dest));

dest.sin\_family = AF\_INET;

dest.sin\_port = htons(atoi(argv[2])); /\* HTTP Server port \*/

if ( inet\_addr(argv[1], &dest.sin\_addr.s\_addr) == 0 )

error\_exit (argv[1]);

/\*---Connect to server---\*/

if ( connect(sockfd, (struct sockaddr\*)&dest, sizeof(dest)) != 0 )

error\_exit ("Connect");

sprintf(buffer, "GET index.html HTTP/1.0\n\n");

send(sockfd, buffer, strlen(buffer), 0);

/\*---While there's data, read and print it---\*/

do

{

bzero(buffer, sizeof(buffer));

bytes\_read = recv(sockfd, buffer, sizeof(buffer), 0);

if ( bytes\_read > 0 )

printf("%s", buffer);

}

while ( bytes\_read > 0 );

/\*---Clean up---\*/

close(sockfd);

return 0;

}

编译C程序

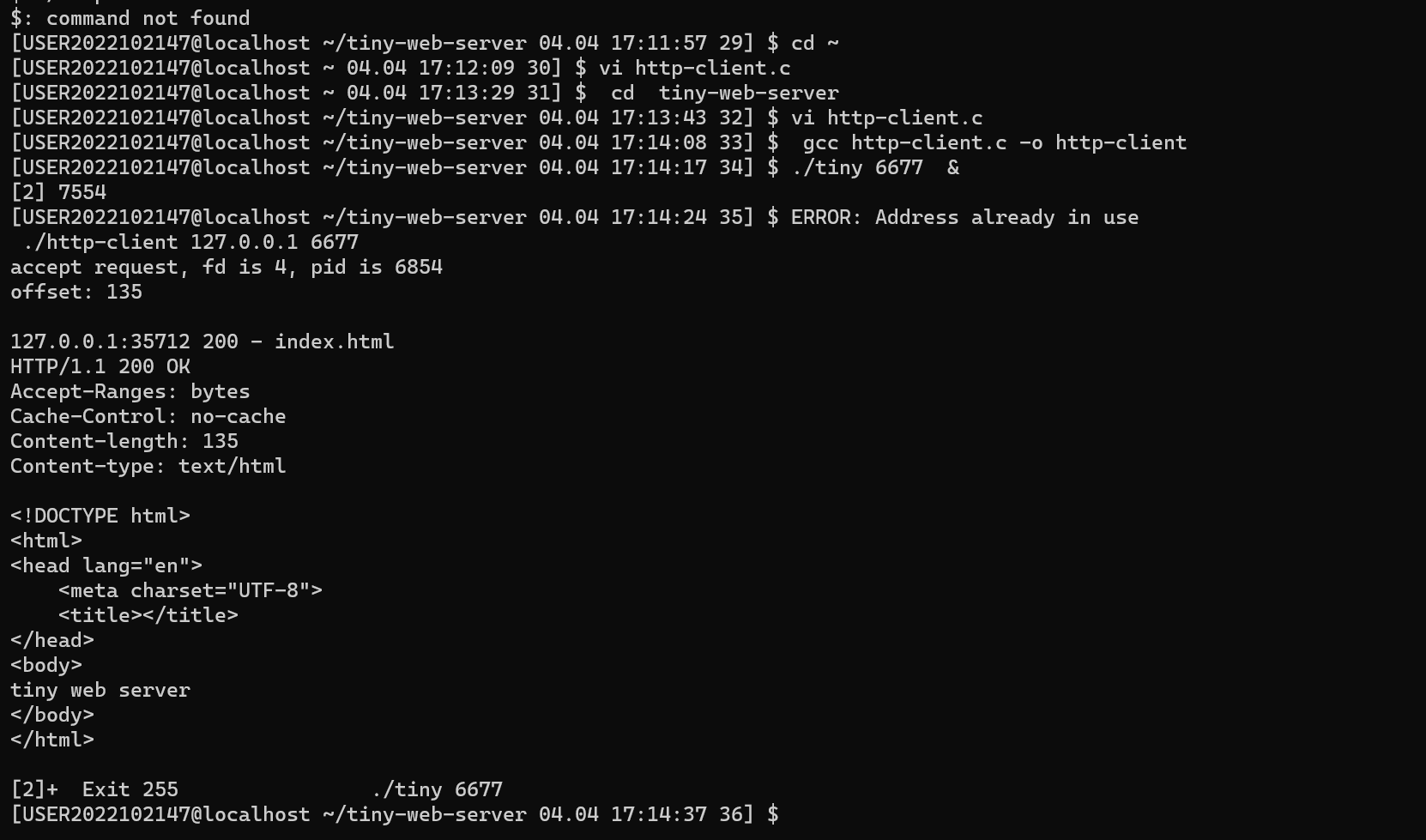
$ gcc http-client.c -o http-client

、启动服务程序；

$ ./tiny 6677 &

运行C客户端

$ ./http-client 127.0.0.1 6677



参考资料：

# [使用 C 语言实现一个 HTTP GET 连接](https://www.cnblogs.com/chorm590/p/c_http_get_201905081356.html)

<https://www.cnblogs.com/chorm590/p/c_http_get_201905081356.html>

11、URL原理、URL编码、URL特殊字符

<https://developer.aliyun.com/article/75516>

解释URL为什么需要编码? 如何编码？

（1）URL需要编码的原因:

1. **非ASCII字符**：例如，中文、阿拉伯文等字符不能直接在URL中使用。
2. **保留字符**：URL中有一些保留字符有特殊用途，如果这些字符用于其他目的，可能会引起歧义。例如，**/**用于分隔不同的路径部分，**?**用于标记查询字符串的开始，**&**用于分隔查询字符串中的不同参数等。
3. **不安全字符**：一些字符可能在某些上下文中不安全，例如，空格、引号、**<**和**>**等，它们可能会被误解为HTML或脚本的一部分，从而可能导致安全问题。
4. **特殊字符**：如空格、**+**等，在URL中有特定的含义或者不被允许使用

（2）

1. **确定需要编码的字符**：首先判断URL中是否含有非ASCII字符、保留字符或不安全字符。
2. **进行编码**：将这些需要编码的字符转换为它们的ASCII码值（对于非ASCII字符，先转换为相应的字节序列，再进行编码），然后转换为十六进制数，并在前面加上**%**符号

12、学习exec。

<https://blog.csdn.net/weixin_50595937/article/details/128714621>

参考以上网页。

#include <stdio.h>

#include <unistd.h>

extern char \*\* environ;

void main(int argc, char \*\* argv)

{

char cc;

printf("in argv[0]%s : \n ", argv[0]);

printf("in argv[1]%s : \n ", argv[1]);

printf("enviro[0]: %s : \n ", environ[0]);

printf("enviro[1]: %s : \n ", environ[1]);

printf("uid=%d, euid=%d, pid=%d, ppid=%d\n",

getuid(),geteuid(),getpid(),getppid());

printf(" end of uid \n ");

}

以上程序命令为 uid.c ,

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

char \* myenv[]={"KEY1=key1","TEST=Value",NULL};

char \* myargv[]={"uid","-l",NULL};

extern char \*\* environ;

void main(int argc, char \*argv[])

{

if (argc<2)

{

printf("Usage: ./exec arg \n");

return;

}

printf("in %s : \n ", argv[0]);

printf("enviro[0]: %s : \n ", environ[0]);

printf("uid=%d, euid=%d, pid=%d, ppid=%d\n",

getuid(),geteuid(),getpid(),getppid());

printf("before exec %s: \n ",argv[1]);

if (execve(argv[1],myargv,myenv) == -1)

perror("execve");

printf("after exec: \n ");

}

以上程序命名为 exec.c ,

请在你的环境中，依次运行以下命令，给出命令的结果 ，并解释命令的功能。

$ gcc uid.c -o uid

$ gcc exec.c -o exec

$ echo $PPID

$ echo $$

$ ./uid argv1

$ ./exec

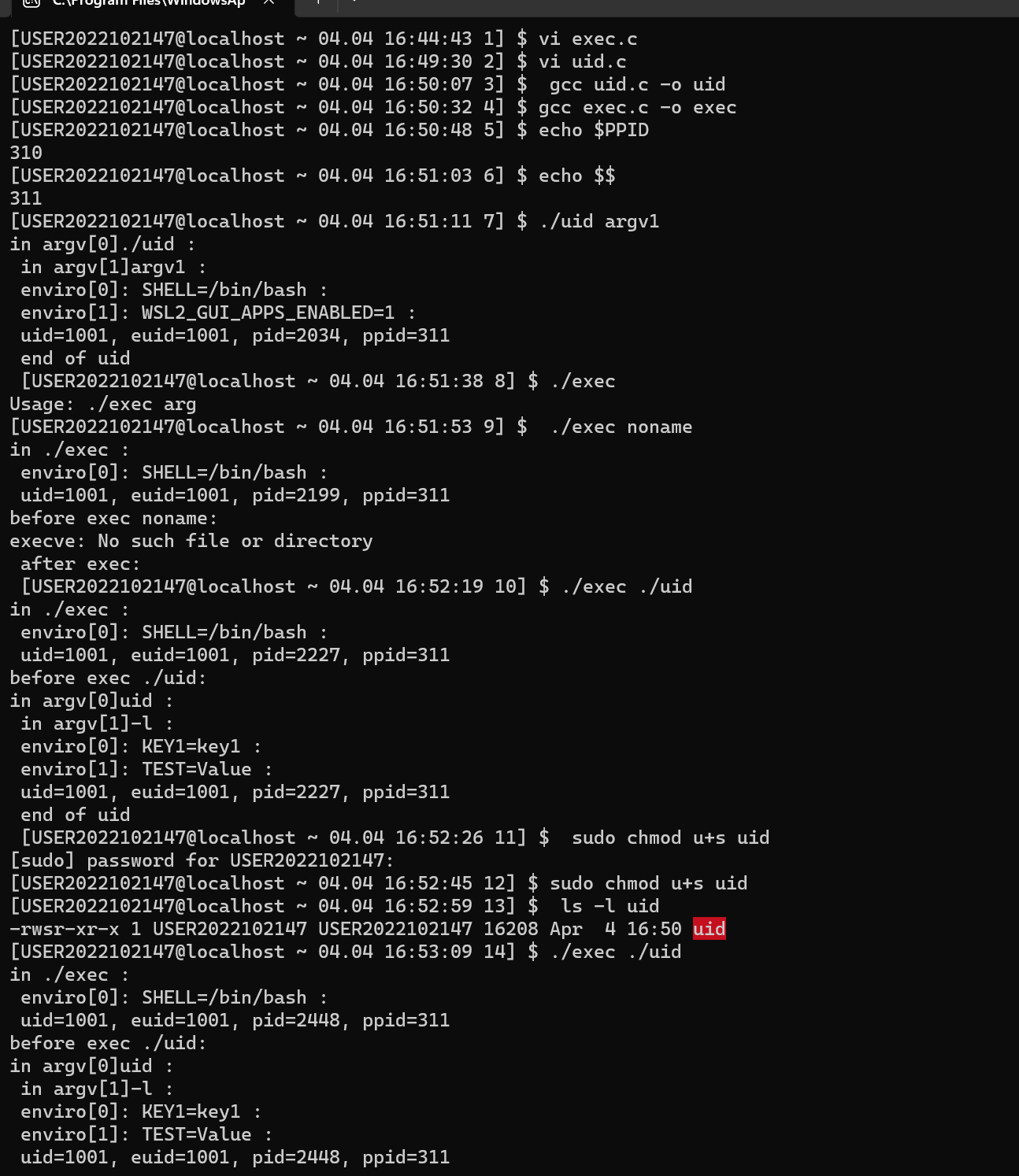
$ ./exec noname

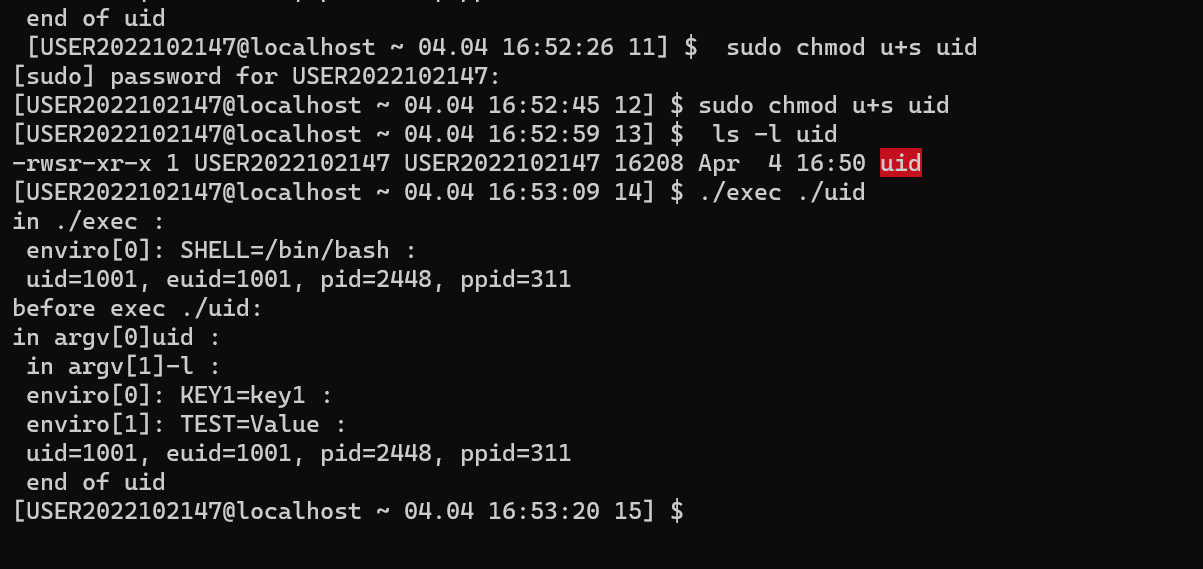
$ ./exec ./uid

$ sudo chmod u+s uid

$ ls -l uid

$ ./exec ./uid





* **编译两个程序**：
  + **gcc uid.c -o uid** 编译第一个程序，生成可执行文件 **uid**。
  + **gcc exec.c -o exec** 编译第二个程序，生成可执行文件 **exec**。
* **获取当前进程信息**：
  + **echo $PPID** 显示当前Shell的父进程ID。
  + **echo $$** 显示当前Shell的进程ID。
* **运行程序 uid**：
  + **./uid argv1** 这将执行 **uid** 程序，其中 **argv1** 是传递给程序的参数。程序会打印传递给它的参数、部分环境变量、UID、EUID、PID和PPID。
* **尝试使用 exec 程序**：
  + **./exec** 由于 **exec** 程序至少需要一个参数才能正常运行，不带参数运行将提示使用方法。
  + **./exec noname** 将尝试执行名为 **noname** 的程序，因为这个文件不存在，**execve** 调用将失败，并打印错误信息。
  + **./exec ./uid** 将使用 **exec** 程序来执行 **uid** 程序。在 **exec** 执行之前和之后的UID、EUID、PID、PPID将会被打印出来。
* **改变 uid 的权限**：
  + **sudo chmod u+s uid** 修改 **uid** 文件，设置 set-user-ID（SUID）权限位。这意味着无论谁运行 **uid** 程序，程序都会以文件所有者的权限执行。
  + **ls -l uid** 显示 **uid** 文件的权限，确认SUID位已被设置。
  + **./exec ./uid** 再次使用 **exec** 程序执行 **uid**。因为设置了SUID，即使是普通用户运行 **exec**，执行 **uid** 时也会拥有 **uid** 文件所有者的权限。这将影响程序中打印的UID和EUID值。

13．掌握c的位运算。

参考资料：<https://www.runoob.com/w3cnote/bit-operation.html>

提示：用&取一个数的指定位，配合<< 或 >>移位操作。

将字符A点阵，由原来的字符’0,、’1’, 改为位0、1, 这样字模的大小由63个字节，减少到9个字节。通过位运算，重写程序，实现字符的显示。

char dot\_A1[9]={0x00,0x08,0x14,0x22,0x41,0x41,0x7f,0x41,0x41}; //16进制，每bit表示一个点。

/\*

char dot\_A[9][8]={

"0000000",

"0001000",

"0010100",

"0100010",

"1000001",

"1000001",

"1111111",

"1000001",

"1000001"

}; \*/

